Memory Efficient Protocols for Detecting Node Replication Attacks in Wireless Sensor Networks

Ming Zhang Vishal Khanapure Shigang Chen Xuelian Xiao
Department of Computer & Information Science & Engineering, University of Florida

***Abstract*—Sensor networks deployed in hostile areas are subject to node replication attacks, in which an adversary compromises a few sensors, extracts the security keys, and clones them in a large number of replicas, which are introduced into the network to perform insider attacks. Memory overhead, energy efficiency and detection probability are the main technical concerns for any replication detection protocol. The previous distributed solutions either require network-wide spontaneous change of pseudo-random numbers or incur significant memory and energy overhead to the sensors, especially in the central area of the deployment. In this paper, we propose four replication detection protocols that have high detection probability, low memory requirement, and balanced energy consumption. The new protocols use Bloom filters to compress the information stored at the sensors, and use two new techniques, called *cell forwarding* and *cross forwarding*, to improve detection probability, further reduce memory consumption, and in the mean time distribute the memory and energy overhead evenly across the whole network. Simulations show that the protocols can achieve nearly 100% detection probability with average memory reduction up to 91%.**

I. Introduction

Security is one of the top design criteria for many sensor networks, particularly for large-scale military deployment involving thousands of sensor nodes that perform critical tasks in hostile areas. These areas are sometimes physically accessible to camouflaged enemies. If an adversary manages to capture a sensor and extract the authentication/encryption keys, it can produce a large number of replicas with the keys and integrate them into the sensor network at chosen locations, which is called the *node replication attack*. Once these replicas gain the trust of other nodes, they can launch a variety of insider attacks [1]. For instance, the replicas may spy for confidential information and leak it to the adversary. It may inject false data to cause an intended bias in the aggregation of the sensors' readings. It may block packets at critical locations. It may collaborate to revoke legitimate nodes. If left undetected, these replicas will cause severe consequences and can even subvert the entire network.

The challenge for detecting node replication attacks stems from the resource scarcity of sensor nodes. An effective solution must be able to detect each occurrence of an authentication key being used at two or more different locations in the network. Such detection requires network-wide comparison of location-dependent authentication information. However, the limited memory (less than 10K RAM for typical low-end sensors that can be used in large quantities [2]) and energy supply place severe constraints on how much authentication information can be stored and exchanged in the network. Therefore, the main performance criteria for replication detection are memory efficiency, energy efficiency, and detection probability. The recent research has been striving for solutions that use less memory and energy.

Parno *et al.* [1] pioneered a Line-Selected Multicast (LSM) solution, which produces a digitally-signed location claim for each sensor and stores the claim at sensors along k random line segments in the network. In normal situations, one key is only used to sign for one location (where the sensor that owns the key resides). However, the replicas have to use the same key to sign for different locations where they reside. It can be shown that, when k is reasonably large, the line segments of two *conflicting location claims* (signed by the same key for different locations) are highly probable to intersect. The node at the intersection will have both claims and thus be able to detect the replication attack.

LSM has a few limitations that remain unsolved to date. For a network of n nodes, each sensor has to store $O(k\sqrt{n})$ location claims on average, which can easily exceed 10K RAM as our analysis in Section IV-B shows. The bigger problem is that random line segments tend to pass the central area of the deployment region more frequently. The nodes there will suffer far worse memory/energy overhead than the average. This is called the *crowded center problem*. Furthermore, we demonstrate in this paper that, even when two line segments intersect, they may not intersect at a common node. In such a case, the replication attack will not be detected. It is called the *cross over problem*.

Conti *et al.* [3] solve the crowded center problem by introducing a network-wide pseudo random number seed that is periodically renewed and must be known instantaneously to all nodes in the network. The infrastructure for distributing such a pseudo-random number seed (such as a satellite or a broadcasting ground station) may not always be available. Other related work that does not use location claims also has their limitations, which will be discussed shortly.

In this paper, we propose four replication detection protocols (B-MEM, BC-MEM, C-MEM and CC-MEM) that are able to detect the replicas with high probability, low memory overhead, and balanced energy consumption. Our first protocol, B-MEM, reduces the number of location claims that each sensor stores from $O(k\sqrt{n})$ to $O(k)$ through the use of two compact Bloom filters. We design a novel mechanism to encode location-claim information in the Bloom filters and exploit this information for replication detection. Next, we propose a new technique called *cell forwarding* to solve the cross over problem, which leads to our second protocol BC-MEM. It improves the detection probability and in the mean time further reduces the memory overhead. Our third protocol C-MEM addresses the crowded center problem by applying another new technique called *cross forwarding* to evenly distribute communication/memory overhead among all nodes in the network. Finally, our last protocol CC-MEM integrates cell forwarding and cross forwarding to achieve the best performance. We evaluate our protocols through extensive simulations. The results show that 1) nearly 100% detection probability can be achieved, 2) memory consumption is greatly reduced — by up to 91% for the average memory requirement and up to 97% for the maximal memory requirement, and 3) energy consumption is balanced — the energy overhead in the central area is cut by up to 47%.

II. Models

A. Network Model

We consider a large sensor network deployed in a hostile environment. Sensor nodes are densely and uniformly deployed in a convex area. They are stationary after deployment. Neighboring nodes form wireless links and data communications between them are protected by preloaded key materials. Each node knows its own geographic location and its neighbors' locations. Many localization schemes [4], [5] can be used to provide such location information. This enables geographic routing [6], [7], which can route a packet hop by hop closer to a given location. The sensors' clocks are synchronized before deployment. Information about location and time is indispensable for critical applications that require sensors to report the position and time of certain events such as where and when an enemy tank is spotted. We expect that modern system clocks carried by the sensors do not drift too much during the operation period. We assume the use of the identity-based key system. By carrying a private key that is derived from a master secret and its ID, each node is able to establish a pairwise secret with any neighboring node for mutual authentication and negotiation of data encryption keys [8], [9]. In addition, a node can make digital signature that is verifiable by other nodes in the network simply based on its ID [10], [11].

B. Threat Model

We assume that the deployment region is neither under the total control of friendly forces nor under the control of the enemy forces. Small groups of camouflaged enemies may operate at certain locations in the region and capture some sensor nodes. But a blanket search of the region is not possible. Otherwise, node replication attacks would not be necessary. After extracting authentication/encryption keys from the compromised nodes, the adversary may launch arbitrary attacks with the keys, including node replication attacks, in which fake nodes that carry the compromised keys may be dropped into the region. We also make the same assumption as in the prior work [1], [3] that any cloned node has at least one legitimate node as a neighbor. This assumption can be removed by using a technique in [1], which probabilistically introduces virtual neighbors (more than one hop away) for each node.

III. Background

A. Distributed Solutions based on Location Claims

In some state-of-the-art solutions for detecting the node replication attacks [1], [3], [12], each node is required to sign its true location with its private key in a *location claim*. The neighbors will verify the signature and the correctness of the claim. If a node refuses to produce such a claim, the neighbors will cut it off by refusing to communicate with it. The location claims will then be stored centrally or distributedly. If it is found that the same private key is used to sign two or more different location claims, we can conclude that the node that owns the private key was compromised and the key was replicated on other malicious nodes. In order to allow new legitimate nodes to join the network at any time and existing nodes to relocate, the above detection process is performed periodically to prevent malicious replicas from mixing in. In each detection period, location claims are generated, stored, and checked for replication.

Applying digital signatures in sensor networks has become feasible [1], [13], thanks to the rapid advance in sensor technologies. The identity-based public key system [10], [11] is used so that each node only stores its *own private key* and a *master public key*. The private key is computed before deployment from the node's ID and a *master private key* (a secret not loaded on any node). For any other node to verify the signature of a location claim, it only needs to compute the public key of the node that produces the claim based on the node's ID and the master public key. Each location claim must carry a node's ID, its location and a signature. Two claims *conflict* if they carry the same ID and their signatures can both be verified correctly, while their locations are different.

Without knowing the master private key, the adversary will not be able to produce a valid pair of ID and private key that can generate a verifiable signature. The only thing he can do is to replicate the compromised private keys, together with the corresponding IDs, in the malicious nodes called *replicas*.

Existing solutions differ in how they store the location claims, which has a huge implication on the memory and communication overhead. The simplest solution is to send all location claims to a base station, which however creates a single point of failure and causes large communication overhead at the nodes around the base station. Moreover, it will not work if the base station stays quiet in a hostile environment except when it needs to read data from the network.

Another simple solution is to broadcast each claim to the entire network so that every node can detect conflicting claims. But the communication overhead will be too high. To reduce the overhead, we may store each claim at a pseudo-random location determined by the ID, and consequently the conflicting claims will be forwarded to the same location for detection. The problem is that, after the adversary compromises a node with a certain ID, it knows the location to which the conflicting claims will go. By jamming that location or compromising the node there, the adversary can produce as many replicas as he wants without being detected [1].

To solve the above problem, one approach [12] is to divide the deployment area into cells and send the location claim of a node to one or several cells that are pseudo-randomly determined by the node's ID. The claim is broadcast to all nodes in the cells but stored by some of them. This approach only partially solves the problem because the cells for each claim can still be determined by an adversary. Moreover, broadcasting in the cells causes considerable communication overhead. Another approach to solve this problem is to spontaneously change the network-wide pseudo random seed at the beginning of each detection period [3]. It assumes that the adversary cannot physically move to a location faster than the conflicting claims do in a short time after the new seed is released. The approach works because the conflicting claims are detected before the adversary can reach the pseudo-random location to which the claims are forwarded. However, spontaneous change of the network-wide pseudo random seed either requires centralized broadcasting from a satellite, a UAV, or a ground station, which may not be always available, or requires expensive distributed mechanisms for leader election and secure seed flooding.

Another solution for guarding the location where a claim will be stored is to make the location unpredictable. Namely, we store each claim at *random, unpredictable* places in the network. This is the approach taken both by [1] and by this paper.

In [1], Parno *et al.* propose two novel solutions that randomize where the claims are stored. Their *randomized multicast* (RM) stores each location claim in $O(\sqrt{n})$ randomly selected *witness nodes*, where n is the number of nodes in the network. The birthday paradox ensures that two conflicting claims have a high probability of sharing a common witness node. This node will detect the attack because it has both claims that use the same key to sign for the same ID at different locations. The node will then broadcast the conflicting claims in the whole network, such that all nodes can verify the claims and refuse to forward packets from the nodes with the ID in the claims. However, each node has to store $O(\sqrt{n})$ claims on average and the total communication overhead is $O(n^2)$ if the average path length is $O(\sqrt{n})$. To reduce the communication overhead, the *line-selected multicast* (LSM) stores a node's claim along several paths (also called *line segments*) from the node to random locations in the network. The basic idea is that the line segments for two conflicting claims are likely to intersect and the node at the intersection can detect the replication because it has both claims. Comparing with RM, LSM significantly reduces communication overhead. However, it also has problems, which are explained below.

B. Memory Overhead Problem

LSM requires each node to store $O(k\sqrt{n})$ claims, where k is the average number of line segments for each claim. To ensure a high probability for the line segments of conflicting claims to intersect, k should be reasonably large (such as six in [1]). The adversary may have supercomputers at remote sites at his disposal to crack the signatures for private keys. Hence, to ensure high-level security, the size of each claim should be large because the strength of a digital signature depends on its length (which is 40 bytes for DSS [14]). Therefore, when n is very large and each sensor has a small memory due to low-cost constraints, $O(k\sqrt{n})$ location claims can present a serious challenge to a sensor's memory space, considering that the sensor has to perform many other measurement/communication/computation/security functions and thus the space allocated for replication detection may be only a small fraction of the available memory.

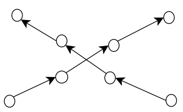


Fig. 1. An example of the cross over problem between two line segments.

C. Crowded Center Problem

It is well known that randomly drawn line segments pass the central area much more frequently than the peripheral area in a convex deployment region [15], [3]. Consequently, the nodes that reside in the central area may experience much higher communication overhead than the nodes near the borders. Moreover, as the location claims are each stored on the nodes along k line segments, a node close to the center will store much more claims than a node close to the edge of the network. In our simulation with 5000 nodes, the space requirement for a center node is more than 14 times that for a peripheral node, and this ratio widens when the network size increases.

D. Cross Over Problem

As illustrated in Figure 1, we observe a subtle "cross over problem" in LSM — two line segments crossing each other do not meet at a real node. If two line segments originating from two replicas do not intersect at a common node, the replicas will escape the detection. To increase the probability of meeting at a common node, more line segments are needed to create more crossing points such that, by chance, at least one of the crossing points happens at a real node. However, this will increase both memory and communication overhead. The cross over problem cannot be solved by overhearing. The energy consumption will be high if a node has to overhear each location claim transmitted in the neighborhood, verify the signature and check for conflict against the locally-stored claims. Note that the energy spent for receiving a packet can be half of the energy needed for transmitting a packet [16].

E. Other Related Work

In [17], in order to defeat the node replication attacks, neighboring nodes establish social fingerprints after deployment and police each other by using the fingerprints during operation. This approach assumes that no node is compromised during the deployment time and no new nodes join or existing nodes relocate after deployment. Replication attacks in mobile sensor networks are addressed in [18], which exploits mobility for replication detection and cannot be applied in static sensor networks. SET [19] requires the base station to periodically coordinate the construction of a tree structure in the network and pull the nodes' IDs up in the tree to check for duplicate identifiers. This approach requires a central coordinator. It also requires each intermediate node in the tree to store the IDs pulled from the subtree rooted at the node.

IV. Memory Efficient Protocols for Replication Detection

In this section, we propose four memory efficient protocols for replication detection. B-MEM uses two Bloom filters to solve the memory overhead problem, such that the average number of location claims stored at each node is reduced from $O(k\sqrt{n})$ to $O(k)$. Designed on top of B-MEM, BC-MEM employs a new *cell forwarding* technique to solve the cross over problem such that when two line segments intersect, they intersect at a common node. Again designed on top of B-MEM, C-MEM employs a new *cross forwarding* technique to solve the crowded center problem such that all nodes experience similar communication overhead. CC-MEM can be viewed as the combination of BC-MEM and C-MEM. It only uses a small fraction of the space required by LSM and yet achieves higher detection probability. It balances the memory usage and energy consumption over the whole network and avoids the hotspot at the center of the network.

A. Preliminaries

1) Location Claim: We denote the location claim of a node α as $C_\alpha = \langle ID_\alpha, l_\alpha, [H(ID_\alpha, l_\alpha)]_{K_\alpha}\rangle$, where ID_α is the node's ID, l_α is the node's location, e.g., in the format of (x, y), K_α is the node's private key (which is used to generate the digital signature), and H is a hash function.

2) Bloom Filter: It is a bit array used for membership test [21]. It encodes a set of members and answers queries about whether a given element is a member in the set. The bits are initialized to zeros. When a new member z is inserted, it is mapped to a number of bit locations in the array through u hash functions, $H_i(z)$, $1 \le i \le u$. To encode the membership of z in the array, those u bits are set to one. To test whether an element z' is a member in the set, we check if the u bits, $H_i(z')$, $1 \le i \le u$, are all ones. If so, z' is in the set; otherwise, it is considered to be not. A Bloom filter has zero *false negative*, meaning that if it answers that an element is not in the set, the element is truly not in the set. The filter however has *false positives*, meaning that if it answers that an element is in the set, the element may not be in the set. According to [22], the probability P_B for a Bloom filter to mistake a non-member z' for a member is

$$P_B = (1-(1-\frac{1}{m})^{su})^u \approx (1-e^{-\frac{su}{m}})^u, \qquad (1)$$

where s is the number of members and m is the number of bits in the array. This probability quickly diminishes when we increase m. All our four protocols use the Bloom filter to reduce memory consumption.

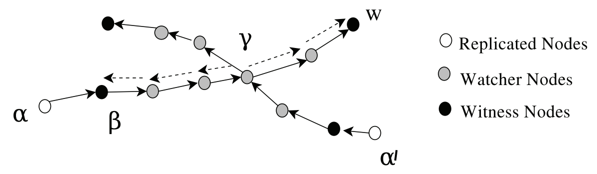


Fig. 2. The solid arrows show how the location claims are forwarded. The dashed arrows show how a conflicting claim C'_α is forwarded during a claim chase initiated by node γ. A witness node, either β or w that stores C_α, will confirm and report the node replication attack once it receives C'_α.

3) Watcher Nodes and Witness Nodes: To verify the legitimacy of a node α, two types of other nodes are involved: the *watcher nodes* and the *witness nodes*. More specifically, as the location claim C_α is forwarded from α to a random place in the network, all intermediate nodes along the routing path (line segment) are watcher nodes, whereas the first node and the last node on the path are witness nodes. While a witness node stores a complete copy of the location claim, a watcher node does not. Instead, it inserts α's ID and location into two Bloom filters, called the *ID filter* and the *location filter*, respectively. Using only a few bits, a watcher node stores the information that it has seen ID_α and l_α. The Bloom filters are initialized to zeros before each detection period begins.

B. *Memory **E**fficient **M**ulticast using **B**loom filters (B-MEM)*

1) Overview: In our first protocol, called B-MEM, the location claim C_α of a node α is multicast via its neighbors to a number of randomly-selected locations in the network. Each neighbor β has a probability p to participate in the multicast. If it does, it becomes a witness node and sends C_α to a random location in the network. The node closest to that location will be another witness node w to store C_α. The watcher nodes on the routing path P from β to w only store the membership of ID_α and l_α in the Bloom filters. Such membership information can help them detect any conflicting location claim C'_α received later, and guide C'_α along P to either β or w, which will then broadcast both C_α and C'_α to the entire network in order to revoke node α and its replicas. Below we describe the protocol in detail.

2) Protocol Details: At the beginning of each detection period, a node α is required to broadcast its location claim C_α to its one hop neighbors; otherwise, neighbors will refuse to communicate with α. Once a neighbor β receives C_α, it checks the correctness of the location l_α and then verifies the signature. After C_α is validated, with probability p, β makes itself a witness node and then picks a random location l_{dest}, to which C_α is forwarded.

When a node γ receives C_α, it performs *two-phase conflict check* to see if conflicting claims can be detected. In phase one, node γ first verifies the signature in C_α; a forged location claim is immediately dropped. It then compares C_α with the locally-stored location claims — node γ serves as witness nodes for others and thus stores some location claims. If γ

finds a claim that conflicts with C_α, it detects a replication attack. It will report the replicas by broadcasting the conflicting claims to the whole network. If γ does not find a claim that conflicts with C_α, it goes to the next phase.

There are two cases in phase two. In the first case, γ finds that ID_α is in its ID filter while l_α is not in its location filter. We say C_α *conflicts* with the Bloom filters. It means a conflicting location claim C'_α that carries ID_α and a different location l'_α has passed γ before and left its footprint in the filters. As shown in Fig. 2, node γ will initiate a *claim chase* by locally broadcasting C_α to its one-hop neighbors. Any neighbor that also finds that C_α conflicts with its Bloom filters will continue the chase with a one-hop local broadcast. If it is a true conflict, the claim chase will eventually lead C_α to the witness node β or w that possesses a complete copy of C'_α. The replication attack is detected.

In the second case, γ finds that ID_α is not in its ID filter or l_α is in its location filter. C_α is regarded as a legitimate location claim, and we say it has *passed* the two-phase conflict check. Node γ tries to find a neighbor closer to l_{dest}. If geographic routing cannot move further to l_{dest}, node γ will act as a witness node and store C_α in its buffer. Otherwise, γ serves as a watcher node. It inserts ID_α and l_α in its two Bloom filters respectively, and then forwards C_α toward l_{dest}.

3) False Alarm, False Verification and Routing Disruption: Suppose node γ receives α's location claim C_α for the first time. Both ID_α and l_α should not be in γ's Bloom filters. However, γ may find that ID_α is in its ID filter due to false positive. Now if l_α is not found in the location filter, γ will regard C_α as a conflicting claim and start the claim chase. This is called a *false alarm*. The probability for a false alarm to happen, denoted as P_f, is

$$P_f = P_B(1-P_B) \approx (1-e^{-\frac{su}{m}})^u(1-(1-e^{-\frac{su}{m}})^u),$$

which can be made very small when we increase m. For example, $P_f = 0.1\%$ in our simulations; see Section V for parameter settings. Even when a false alarm occurs, the only overhead is for C_α to travel one hop in the claim chase. The probability for the claim chase to continue for r hops is $(P_f)^r$, which diminishes quickly in r. We stress that a false alarm never leads to the misreport of a node replication attack, which requires two digitally-signed conflicting claims to be actually present at a node at the same time.

Now suppose node γ receives a true conflicting claim C_α. It should detect that ID_α is a member of the ID filter but l_α is not a member of the location filter, meaning that it has seen ID_α before but with a different location. However, a false positive in the location filter may announce that l_α is in the filter even though it is not. This causes a *false verification*, in which node γ mistakenly regards C_α as legitimate. The probability for a false verification to happen is P_B, given in (1), which can be made exceedingly small. Moreover, since line segments of two replicas may intersect at many (say, r') nodes, the probability for a replication attack to evade detection in one detection period is $(P_B)^{r'}$. Even if this happens, the attack will be detected with probability $(1-(P_B)^{r'})$ in each subsequent period (because the line segments that a location claim travels in each detection period are different).

The replicas, after they are introduced into the network and before they are detected and excluded from the network, may try to disrupt the detection. The erratic behavior of the replicas will be guarded by the surrounding neighbors that will cut them off from the network if they operate abnormally. However, the replicas may try to disrupt routing by dropping the received location claims or replacing them with forged ones that will be dropped by the next-hop nodes due to the failure of signature verification. Dropping legitimate claims does not cause any problem. However, dropping location claims that originate from other replicas will reduce the lengths of the corresponding line segments. The shortened line segments are less likely to intersect. On the other hand, more replicas mean more line segments, which increases the chance of intersecting. Combining these two factors, our simulations in Section V-B show that *a larger number of replicas (that drop instead of forwarding location claims) will actually increase the probability for B-MEM to detect the node replication attack*. The same thing is true for other protocols in this section.

4) Analysis: In LSM [1], each node stores $O(k\sqrt{n})$ location claims on average. Hence, the memory complexity is $O(tk\sqrt{n})$, where k is the average number of line segments for each claim and t is the size of a location claim. Suppose $k=6$, the average length of a line segment is 50 (which is the $O(\sqrt{n})$ component in the memory complexity), and t is 46 bytes (2 bytes for ID, 4 bytes for location coordinates, and 40 bytes for digital signature, the same as DSS). Then 13.8K bytes are required on average for each node, which may exceed the total RAM size of many sensors (4K $\sim$ 8K bytes [2]).

In B-MEM, each node only stores $2k$ location claims on average. The size of a Bloom filter is $O(t'k\sqrt{n})$, where t' is the number of bytes that a Bloom filter uses to record the membership of an element. Hence, the memory complexity is $O(tk + t'k\sqrt{n})$. For example, if $t' = 2$ bytes (that ensures $P_B < 0.1\%$) and the other parameters are the same as above, then only 1.15K bytes are needed for each node.

In terms of communication overhead, the number of messages sent and received by all nodes in the network is $O(kn\sqrt{n})$ for both LSM and B-MEM.

*C. **M**emory **E**fficient **M**ulticast using **B**loom filters and **C**ell forwarding (BC-MEM)*

1) Overview: BC-MEM is designed on top of B-MEM. It adopts a *cell forwarding* technique that not only solves the cross over problem (Figure 1) but also reduces the memory overhead. We divide the deployment area into virtual cells. In each cell, we assign an *anchor point* for every node in the network. The anchor point for a node α is determined by α's ID. The node closest to the anchor point is called the *anchor node* for α. Recall that when B-MEM forwards a location claim on a line segment, all intermediate nodes on the line serve as watchers, while the first node and the last node serve

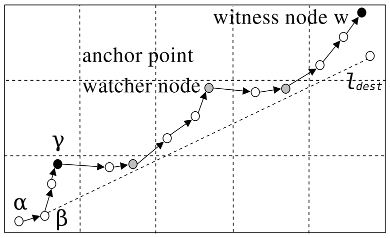


Fig. 3. BC-MEM forwards a location claim through anchor points in the cells that the line segment intersects.

as witnesses. In contrast, BC-MEM does not forward a claim on the line segment. It forwards the claim to the anchor point in the next cell that the line segment intersects. As shown in Figure 3, the claim is forwarded from one anchor node to another until reaching the last cell. The anchor nodes in the intermediate cells are watchers, and the anchor nodes in the first and last cells are witnesses.

2) Protocol Details: We only describe the part of the protocol that is different from B-MEM. Every node is pre-configured with the cell size and the coordinates of a base point (x^*, y^*) that serves as the bottom-left corner of one cell. The cell size and the base point together uniquely define the cell structure of the whole network. Based on its own coordinates, each node knows which cell it belongs to. Consider an arbitrary cell whose bottom-left corner has coordinates (x_0, y_0). Without losing generality, let the cell size be 1×1. The anchor point (x_α, y_α) in the cell for node α is defined as

$$x_\alpha = x_0 + H'(ID_\alpha|i), \qquad y_\alpha = y_0 + H'(ID_\alpha|i), \qquad (2)$$

where H' is a hash function whose range is $[0, 1)$. Parameter i is optional in the formula. It is the detection period number that is increased by one after each period. Its purpose is to make the anchor points different over time.

When a neighbor β of node α decides to forward C_α to a random location l_{dest}, it first computes the anchor point for node α in the cell where it resides. It forwards C_α towards that point. When the node γ closest to that point receives C_α, it cannot forward C_α any further. The node will designate itself as a witness node and perform the two-phase conflict check as described in Section IV-B. If C_α passes the check, γ will compute the anchor point for α in the next cell that the line segment intersects and forward C_α towards that point. The node closest to that point will perform the two-phase conflict check and then become a watcher node before forwarding C_α to the next cell that the line segment intersects. When C_α reaches the last cell that intersects with the line segment, the anchor node in that cell becomes a witness and stores C_α.

If C_α triggers the claim chase at a watcher node, the node will forward C_α to the anchor points in the four adjacent cells. After receiving C_α, if any of the nodes closest to those anchor points finds that C_α conflicts with its Bloom filters, it will repeat the process by forwarding C_α to the anchor points in its adjacent cells except for the cell from which C_α was received. Eventually, C_α will reach a witness node and result in the detection of the node replication attack.

3) Discussion: Suppose the line segments of two conflicting location claims, C_α and C'_α, intersect at a point (x, y). Both claims carry the same ID. BC-MEM tries to forward both claims to the same anchor node in the cell where (x, y) resides, which addresses the cross over problem. The memory overhead is also reduced because in each cell only one node serves as a watcher (or witness). Each node is mapped to a different set of anchor points that are pseudo-randomly generated. Therefore, all nodes in a cell have equal chance to serve as watchers and witnesses. The anchor points for a node are distributed across the entire network, and it may change over time according to (2), which makes it extremely difficult for an adversary to capture all nodes that may serve as the watchers or witnesses.

4) Analysis: For B-MEM, the average length of a line segment is $O(\sqrt{n})$ nodes. For BC-MEM, the average length of a line segment is $O(\sqrt{n'})$ cells. Hence, the memory complexity of BC-MEM is $O(tk + t'k\sqrt{n'})$, comparing with $O(tk + t'k\sqrt{n})$ for B-MEM, where n' is the number of cells in the network. It is smaller than the number of nodes, n.

The energy overhead is increased because a location claim is not forwarded along a straight line but instead along a zig-zag path. However, since it solves the cross over problem, BC-MEM may use a smaller number of line segments in order to achieve a certain detection probability, which compensates for the increase in energy overhead due to zig-zag paths.

D. *Memory Efficient Multicast using Cross forwarding (C-MEM)*

1) Overview: Designed on top of B-MEM, C-MEM incorporates a new *cross forwarding* technique to solve the crowded center problem. Recall that B-MEM stores the information about a location claim along randomly selected line segments, which are likely to pass the center area of the deployment. C-MEM first selects a random point (called the *cross point*) in the network. It forwards the location claim to that point. From there, it forwards the claim along the horizontal and vertical lines that pass the cross point. While the node closest to the cross point is a witness node, the nodes along the horizontal and vertical lines are watchers. Since the cross points for all location claims are distributed uniformly at random in the network, it is no longer true that the lines pass the center area more frequently. Note that C-MEM does not use cell forwarding.

2) Protocol Details: First, we describe a simple local coordination mechanism to elect a neighbor of node α, who will be responsible for selecting the cross point. When the network is deployed, each neighbor β of node α sets a *backoff counter* to a random number, denoted as b_β. When β receives C_α, it waits for b_β units of time, which should be much smaller than the length of a detection period. During this time, if node β hears the announcement from another neighbor of α that it will pick the cross point, β will stop waiting and reduce b_β

by one. If β does not hear from anyone within b_β time, it will make the announcement by a two-hop broadcast to cover all α's neighbors, and then randomly pick a cross point to forward C_α. If β does not get the chance to pick a cross point for a number of detection periods, its backoff counter b_β will eventually be reduced to zero, and in this case, β will immediately elect itself in the next period and then reset b_β to another random value.

If β forwards C_α to the cross point, β and the node w closest to that point are witnesses nodes, and all other nodes on the path are watchers. Node w will then forward C_α in four directions representing a horizontal line and a vertical line passing the cross point. Nodes along the lines are watcher nodes. Furthermore, C-MEM uses local multicast to alleviate the cross over problem. When a watcher node γ forwards C_α to its next hop node γ' on the horizontal or vertical line, one or multiple nodes between γ and γ' are randomly selected to receive C_α. These nodes only need to do the two-phase conflict check and do not further forward the location claim.

In a convex deployment area, two sets of crossing lines are likely to intersect at one or two locations, where the replication attack is detected. The most important contribution of cross forwarding in C-MEM is that because the crossing points and the corresponding lines are uniformly distributed in the network, the memory and communication overhead are also uniformly distributed among all nodes.

3) Analysis: In C-MEM, each location claim has only two witnesses. Hence, each node will store two location claims on average. The watcher nodes are located along two lines that cut across the network horizontally and vertically, as well as along the line segment from a neighbor to the cross point. Hence, the memory complexity for each node is $O(2t + 3t'\sqrt{n}) = O(t + t'\sqrt{n})$, which is lower than that of B-MEM.

E. *Memory **E**fficient **M**ulticast using **C**ross and **C**ell forwarding (CC-MEM)*

1) Overview: CC-MEM combines cross forwarding and cell forwarding to solve both the crowded center problem and the cross over problem, such that it can detect node replication attacks with high probability and low overhead.

2) Protocol Details: Similar to BC-MEM, CC-MEM divides the network into virtual cells. A node α has an anchor point in each cell. Similar to C-MEM, after node α broadcasts C_α to its one-hop neighbors, the local coordination mechanism is executed and one neighbor β is elected to start cross forwarding. Node β randomly selects a cell and uses the anchor point for α in the cell as the cross point. It then sends C_α to that point through cell forwarding. The node w closest to the cross point, as well as the anchor node in the cell where α resides, are chosen as witness nodes. In cross forwarding, node w identifies the four adjacent cells and forwards C_α to the anchor points in those cells. The nodes closest to the anchor points will act as watchers. Each watcher forwards the received location claim further along the horizontal or vertical line, cell by cell, to the edge of the deployment area. In each cell, only the anchor node performs the two-phase conflict check and serves as a watcher.

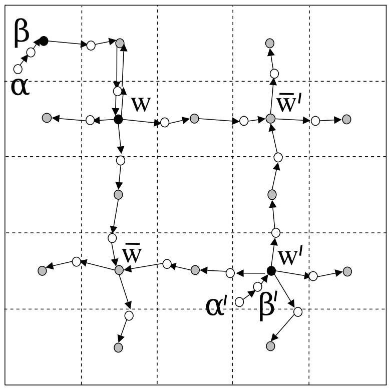


Fig. 4. An example of CC-MEM, where the location claims of two replicas, α and α', are each forwarded along two lines forming a cross.

Figure 4 shows an example of CC-MEM. Two replicas α and α' are located in different places. For node α, its neighbor β is elected to initiate the cross forwarding. Node β sends C_α to a randomly selected cross point. The witness w, which is closest to the cross point, stores a complete copy of C_α in its buffer. From w, the location claim travels in four directions to the borders. Along the way, C_α is forwarded to α's anchor points in the cells it passes, and the nodes closest to those anchor points are watcher nodes, which insert ID_α and l_α to their Bloom filters. The same thing happens to C'_α along two different lines intersecting at w', which is the witness node for α'. The lines of the conflicting claims intersect at two watcher nodes $\bar{w}$ and $\bar{w}'$. After querying their Bloom filters, both $\bar{w}$ and $\bar{w}'$ will detect the replication attack. They will initiate the claim chase by sending C'_α (or C_α) to neighboring watcher nodes. Eventually, the witness nodes w (or w') will receive the conflicting location claims and revoke the two replicas.

3) Discussion: CC-MEM combines cross forwarding and cell forwarding. As we have discussed in the previous two subsections, the former solves the crowded center problem and the latter solves the cross over problem. Together, they are able to achieve high detection probability with low overhead that is evenly distributed across the whole network.

4) Analysis: In CC-MEM, each location claim has two witnesses. Hence, each node will store two location claims on average. Due to cell forwarding, there are $O(\sqrt{n'})$ watcher nodes on each of the three lines that a location claim traverses. Therefore, the memory complexity is $O(t + t'\sqrt{n'})$ for each node, which is the smallest one among our four protocols.

V. Simulation Results

A. *Simulation Setup*

The simulator is developed in C++. In the simulations, n nodes are deployed uniformly at random in a 1000×1000 area, where n varies from 1000 to 10000. We assume that each node has bidirectional communication links with its one-hop neighbors, and select the transmission range such that each node has approximately 20 neighbors. We implement a simplified geographic routing protocol [7] to forward location claims. A node always greedily forwards a location claim to the neighbor closest to the destination. If a node has no neighbor closer to the destination, it stops forwarding.

We implement five replication detection protocols: LSM [1], B-MEM, BC-MEM, C-MEM, and CC-MEM. All protocols are implemented at the application layer. We divide the network into 10×10 virtual cells for cell forwarding in BC-MEM and CC-MEM. We use six line segments for LSM (according to the original paper [1]) and B-MEM, and five line segments for BC-MEM.[1]

In most simulations, we introduce one replica by randomly compromising a node and inserting a replica at a random location. We then run the protocols to collect data for detection probability, memory overhead and energy overhead. For each network size, we run the protocols on 200 different random networks and average the simulation results. We also study the case of multiple replicas in Table II.

B. Detection Probability

First, we run simulations to measure the *detection probability*, which is the probability P_d for a protocol to detect the node replication attack in *one detection period*. If we consider l consecutive detection periods, the probability becomes $(1 - (1 - P_d)^l)$, which approaches to one as l increases no matter how big P_d is. The expected number of detection periods that elapse before the replication attack is identified is $\frac{1}{P_d}$. Hence, a greater value of P_d means that we can detect the attack quicker and thus limit its damage.

Table I shows the detection probabilities of all five protocols under different network sizes. B-MEM has a slightly lower detection probability than LSM in some cases due to false positive of Bloom filters. BC-MEM achieves a higher detection probability than both LSM and B-MEM by using the cell forwarding technique to solve the cross over problem. With fewer line segments, C-MEM's performance is comparable to BC-MEM's because two sets of crossing lines have a very high probability to intersect at one or two locations. Finally, CC-MEM achieves the highest and nearly perfect detection probability because it combines the benefit of both cross forwarding and cell forwarding.

Next, we increase the number of replicas. Each replica will drop the received location claims in order to disrupt the execution of the replication detection protocol. Will more replicas drive down the detection probability because more location claims are dropped? This issue has been discussed in Section IV-B3. The simulation results in Table II show the contrary: more replicas will actually increase the detection probability. For networks of 5000 nodes, the detection probabilities for all protocols improve to one with merely three (or more) replicas. The reason is that more replicas produce more line segments, which have a positive effect of increasing the probability for two line segments to intersect. This positive effect compensates for the negative effect of claim drops.

[1]Since BC-MEM employs cell forwarding to solve the cross over problem, even with one fewer line segment it can achieve higher detection probability than LSM and B-MEM.

TABLE I
DETECTION PROBABILITY IN A UNIFORMLY DEPLOYED SQUARE AREA

	LSM	B-MEM	BC-MEM	C-MEM	CC-MEM
1000	0.89	0.86	0.93	0.95	0.99
2000	0.89	0.86	0.93	0.93	0.98
3000	0.81	0.86	0.97	0.93	1
4000	0.86	0.81	0.96	0.89	0.99
5000	0.86	0.83	0.93	0.95	1
6000	0.80	0.82	0.96	0.95	1
7000	0.82	0.82	0.86	0.90	1
8000	0.78	0.83	0.95	0.91	1
9000	0.83	0.86	0.92	0.91	1
10000	0.81	0.76	0.90	0.92	1

TABLE II
DETECTION PROBABILITY IN A UNIFORMLY DEPLOYED SQUARE AREA WITH DIFFERENT NUMBER OF REPLICAS

	LSM	B-MEM	BC-MEM	C-MEM	CC-MEM
1	0.86	0.83	0.93	0.95	1
3	1	1	1	1	1
5	1	1	1	1	1
10	1	1	1	1	1
100	1	1	1	1	1

C. Memory Overhead

Memory is a critical resource for wireless sensor nodes. Many sensors have no more than a few thousand bytes of memory [2]. Considering that the memory must be shared for other applications, the space allocated for detecting replicas will be very limited. In LSM, a node stores a complete copy of each location claim it receives. Some nodes may have to store several hundred location claims, which will exhaust their memory space. In our protocols, a node exploits Bloom filters to record the footprint of most location claims it receives, and it only stores a few complete claims. This dramatically reduces the memory overhead.

In the simulations, we assume that the size of each location claim is 46 bytes (2 bytes for ID, 4 bytes for location coordinates, and 40 bytes for digital signature as required by DSS). We allocate 15 bits ($m/s = 15$) and use 7 hash

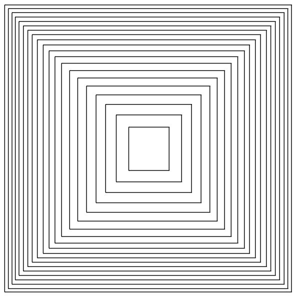
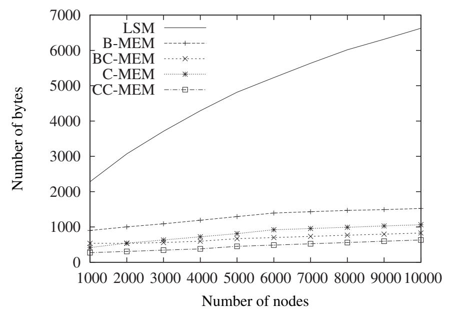
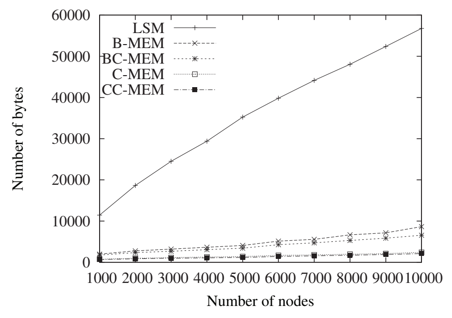


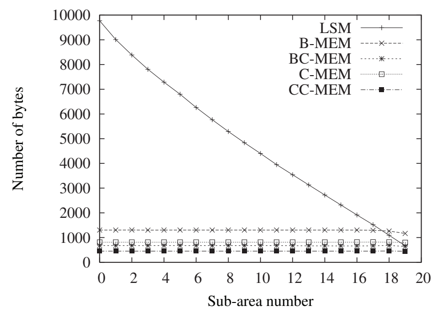
Fig. 5. The deployment region is divided into 20 sub-areas. Each accounts for 5% of the deployment region. *The central sub-area is numbered 0 and the most external sub-area is numbered 19.*



(a) Average memory consumption per node.

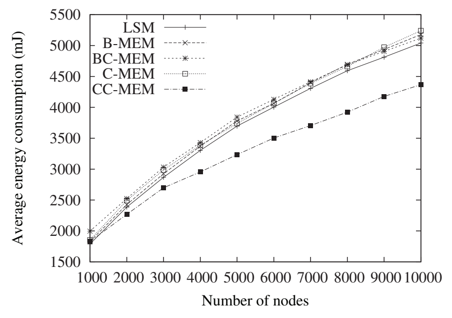


(b) Maximal memory consumption.

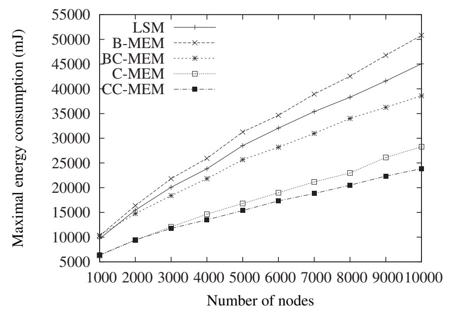


(c) Average memory consumption per node in each sub-area.

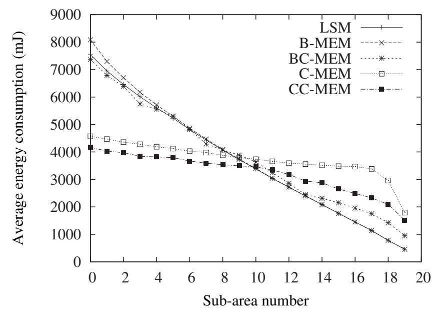
Fig. 6. Memory Overhead. (a) Our protocols significantly reduce the average memory consumption per sensor, when comparing with LSM. (b) The maximal memory consumption is the largest memory usage incurred at any sensor during a protocol's execution. When comparing with LSM, our protocols reduce the maximal memory consumption by a wide margin. (c) The memory overhead is evenly distributed among all sub-areas in our protocols, whereas in LSM it is concentrated in the central area.



(a) Average energy consumption per node.



(b) Maximal energy consumption.



(c) Average energy consumption per node in each sub-area.

Fig. 7. Energy Overhead. (a) CC-MEM has the least average energy consumption. (b) CC-MEM reduces the maximal energy consumption by up to 47% when comparing with LSM. (c) C-MEM and CC-MEM balance the energy consumption across the whole network, whereas LSM, B-MEM, and BC-MEM consume much more energy in the central sub-area.

functions ($u = 7$) to store an element in a Bloom filter. We adjust the size of the Bloom filter according to the network size.

Figure 6(a) shows *the average memory consumption per sensor node*, which is the total memory usage of all sensors divided by the number of sensors. As the network size increases, the average memory consumption also increases because each node needs to store information of more location claims. Compared with LSM, B-MEM reduces the average memory consumption by up to 77% due to the use of Bloom filters. C-MEM reduces the average memory consumption by up to 84% since fewer location claims are stored. Each location claim is stored at 12 witness nodes in B-MEM, while it is stored at two nodes in C-MEM. Both BC-MEM and CC-MEM employ cell forwarding, in which location claims are inserted in the Bloom filters at much fewer watcher nodes. Consequently, the size of the Bloom filters can be reduced. Hence, compared with LSM, BC-MEM and CC-MEM reduce the average memory consumption by up to 87% and 91%, respectively.

In LSM, due to the crowded center problem, some nodes have to store much more location claims than others. We define *the maximal memory consumption* to be the largest memory usage incurred at any sensor during a protocol's execution. As shown in Figure 6(b), the maximal memory consumption in LSM exceeds 10K bytes for all network sizes. It even reaches 56K bytes in networks of 10,000 nodes, whereas our protocols have much smaller maximal memory consumption. Specifically, B-MEM, BC-MEM, C-MEM, and CC-MEM reduce maximal memory consumption by up to 88%, 90%, 96%, and 97% respectively, when comparing with LSM. It is interesting to note that although C-MEM has higher average memory consumption than BC-MEM, it wins the maximal memory consumption comparison because its cross forwarding technique evenly distributes the memory overhead among all nodes.

Next, we study the memory consumption distribution across the whole network. As shown in Figure 5, we divide the network into 20 equal sub-areas from the center to the edges. Each sub-area accounts for 5% of the network. These sub-areas are numbered from 0 to 19 starting from the central sub-area. For networks of 5,000 nodes, we count the average memory consumption per node in each sub-area. The results are plotted in Figure 6(c). It is clear that LSM consumes much more memory in the central sub-area than in the external sub-areas, while our protocols distribute memory overhead much more evenly across the network. The reason is that all nodes in the line segments of LSM are witness nodes and they are crowded in the center of a network, whereas the witness nodes in our protocols are randomly distributed. Although the watcher nodes in B-MEM and BC-MEM are located in the central sub-area more frequently, their Bloom filters are compact and do not incur too much overhead.

D. Energy Overhead

We measure the amount of energy that each sensor node spends in sending and receiving messages. We use the energy model in [16], where the total available energy of a node is 324,000 mJ, sending a bit costs 0.059 mJ, and receiving a bit costs 0.028 mJ.

Figure 7(a) presents the *average energy consumption* per sensor node, which is the total energy expenditure by all sensors divided by the number of sensors. The figure shows that three of our protocols, B-MEM, BC-MEM and C-MEM, consume a little more energy (less than 3% on average) than LSM, which should be acceptable considering their significant memory reduction. Our best protocol, CC-MEM, reduces the average energy consumption by up to 15% when comparing with LSM. This is because CC-MEM uses fewer line segments than LSM, thanks to cross forwarding. Although C-MEM also employs cross forwarding, it consumes more energy because its local multicast requires additional communication overhead.

We define the *maximal energy consumption* as the largest energy expenditure by any node during a protocol's execution. In LSM, due to the crowded center problem, the nodes residing in the central sub-area experience much higher communication overhead than the nodes in the peripheral sub-areas. As shown in Figure 7(b), by exploiting the cross forwarding technique, C-MEM and CC-MEM significantly reduce the maximal energy consumption. In particular, CC-MEM reduces the maximal energy consumption by up to 47% when comparing with LSM. In Figure 7(c), we adopt the same sub-area approach as used in the previous subsection to measure the energy consumption distribution. The figure shows that the energy consumption in LSM increases when we move from the border sub-area to the central sub-area, while C-MEM and CC-MEM are able to balance the energy consumption across the whole network among most sub-areas.

VI. Conclusion

Sensor networks are vulnerable to node replication attacks. In this paper, we propose four distributed protocols for detecting these malicious attacks. The new protocols improve the state of the art by significantly reducing the amount of memory space needed, by balancing the memory and energy consumption across the network, and by improving the detection probability to nearly 100%. The proposed protocols also have a couple of limitations. They cannot detect the replication attacks in a mobile sensor environment. They rely on the relatively expensive public key cryptography. Our future work is to design replication detection protocols that use the secret key cryptography and work for both static and mobile sensors.

VII. Acknowledgments

This work was supported in part by the US National Science Foundation under grant CPS-0931969. We thank our shepherd, Dr. Sonia Fahmy, and the anonymous reviewers for their constructive comments.

References

[1] B. Parno, A. Perrig, and V. D. Gligor, "Distributed Detection of Node Replication Attacks in Sensor Networks," In *Proc. of IEEE Symposium on Security and Privacy (S&P)*, May 2005.
[2] "Sensor Node," *http://en.wikipedia.org/wiki/Sensor_node*.
[3] M. Conti, R. D. Pietro, and L. V. Mancini, "A Randomized, Efficient, and Distributed Protocol for the Detection of Node Replication Attacks in Wireless Sensor Networks," In *Proc. of ACM MobiHoc*, September 2007.
[4] T. Eren, D. Goldenberg, W. Whitley, Y. Yang, S. Morse, B. Anderson, and P. Belhumeur, "Rigidity, Computation, and Randomization of Network Localization." In *Proc. of IEEE INFOCOM*, March 2004.
[5] D. Goldenberg, A. Krishnamurthy, W. Maness, Y. R. Yang, A. Young, A. S. Morse, A. Savvides, and B. Anderson, "Network Localization in Partially Localizable Networks," In *Proc. of IEEE INFOCOM*, March 2005.
[6] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia, "Routing with Guaranteed Delivery in Ad Hoc Wireless Networks," In *Proc. of International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DialM)*, August 1999.
[7] B. Karp and H. T. Kung, "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks," In *Proc. of ACM MobiCom*, August 2000.
[8] D. Boneh and M. Franklin, "Identity-Based Encryption from the Weil Pairing," In *Proc. of CRYPTO*, August 2001.
[9] P. Barreto, H. Kim, B. Bynn, and M. Scott, "Efficient Algorithms for Pairing-Based Cryptosystems," In *Proc. of CRYPTO*, August 2002.
[10] C. Cocks, "An Identity Based Encryption Scheme based on Quadratic Residues," In *Proc. of IMA International Conference on Cryptography and Coding*, pp. 360–363, 2001.
[11] A. Shamir, "Identity-Based Cryptosystems and Signature Schemes," In *Proc. of CRYPTO*, 1985.
[12] B. Zhu, V. G. K. Addada, S. Setia, S. Jajodia, and S. Roy, "Efficient Distributed Detection of Node Replication Attacks in Sensor Networks," *Annual Computer Security Applications Conference (ACSAC)*, December 2007.
[13] D. Malan, M. Welsh, and M. Smith, "A Public-Key Infrastructure for Key Distribution in TinyOS based on Elliptic Curve Cryptography," In *Proc. of IEEE Conference on Sensor and Ad Hoc Communications and Networks (SECON)*, October 2004.
[14] "Digital Signature Standard," *FIPS PUB 186-3*, March 2006.
[15] L. Popa, A. Rostamizadeh, R. Karp, C. Papadimitriou, and I. Stoica, "Balancing Traffic Load in Wireless Networks with Curveball Routing," In *Proc. of ACM MobiHoc*, September 2007.
[16] A. Wander, N. Gura, H. Eberle, V. Gupta, and S. C. Shantz, "Energy Analysis of Public-key Cryptography for Wireless Sensor Networks," In *Proc. of Annual IEEE International Conference on Pervasive Computing and Communications (PERCOM)*, March 2005.
[17] K. Xing, F. Liu, X. Cheng, and D. H. C. Du, "Real-Time Detection of Clone Attacks in Wireless Sensor Networks," *The International Conference on Distributed Computing Systems (ICDCS)*, June 2008.
[18] C. M. Yu, C. S. Lu, and S. Y. Kuo, "Mobile Sensor Network Resilient Against Node Replication Attacks," In *Proc. of IEEE Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, June 2008.
[19] H. Choi, S. Zhu, and T. Laporta, "SET: Detecting Node Clones in Sensor Networks," *International ICST Conference on Security and Privacy in Communication Networks (SecureComm)*, September 2007.
[20] R. Brooks, P. Y. Govindaraju, M. Pirretti, N. Vijaykrishnan, and M. T. Kandemir, "On the Detection of Clones in Sensor Networks Using Random Key Predistribution," *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, vol. 37, no. 6, pp. 1246–1258, November 2007.
[21] B. Bloom, "Space/Time Trade-Offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, July 1970.
[22] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: a Survey," *Internet Mathematics*, vol. 1, no. 4, pp. 636–646, June 2002.